



UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/939,232	08/24/2001	William Joseph Armstrong	IBM / 182	4082

26517 7590 10/31/2006

WOOD, HERRON & EVANS, L.L.P. (IBM)
2700 CAREW TOWER
441 VINE STREET
CINCINNATI, OH 45202

EXAMINER

ART UNIT PAPER NUMBER

DATE MAILED: 10/31/2006

Please find below and/or attached an Office communication concerning this application or proceeding.



UNITED STATES PATENT AND TRADEMARK OFFICE

Commissioner for Patents
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
www.uspto.gov

MAILED

NOV 01 2006

Technology Center 2100

**BEFORE THE BOARD OF PATENT APPEALS
AND INTERFERENCES**

Application Number: 09/939,232
Filing Date: August 24, 2001
Appellant(s): ARMSTRONG ET AL.

Douglas A. Scholer (Registration Number 52,197)
For Appellant

EXAMINER'S ANSWER

This is in response to the appeal brief filed July 19, 2006 appealing from the Office action mailed February 15, 2006.

(1) Real Party in Interest

A statement identifying by name the real party in interest is contained in the brief.

(2) Related Appeals and Interferences

A statement identifying the related appeals and interferences, which will directly affect or be directly affected by or have a bearing on the decision in the pending appeal is contained in the brief.

(3) Status of Claims

The statement of the status of claims contained in the brief is correct.

(4) Status of Amendments After Final

The appellant's statement of the status of amendments after final rejection contained in the brief is correct.

(5) Summary of Claimed Subject Matter

The summary of claimed subject matter contained in the brief is correct.

(6) Grounds of Rejection to be Reviewed on Appeal

The appellant's statement of the grounds of rejection to be reviewed on appeal is correct.

(7) Claims Appendix

The copy of the appealed claims contained in the Appendix to the brief is correct.

(8) Evidence Relied Upon

5,404,563	Green et al.	4-1995
-----------	--------------	--------

5,872,963	Bitar et al.	2-1999
-----------	--------------	--------

Abraham Silberschatz and Peter Baer Galvin; "Operating System Concepts"; John Wiley & Sons, Inc.; Fifth Edition; pp. 74-75

(9) Grounds of Rejection

The following ground(s) of rejection are applicable to the appealed claims:

Claims 1-19 and 21 are rejected under 35 U.S.C. § 103(a) as being unpatentable over US Patent No. 5,404,563 to Green et al. (Green) in view of US Patent No. 5,872,963 to Bitar et al. (Bitar), and further in view of “Operating System Concepts, Fifth Edition” by Abraham Silberschatz and Peter Baer Galvin (Silberschatz).

Green provides background knowledge of what is known in the art regarding a hypervisor on a logically partitioned computer system (column 1, lines 10-54). In particular, Green teaches, “The hypervisor 112 schedules, or allocates, the physical hardware components 104 to the logical partitions 114. For example, during a particular time-slice, the hypervisor 112 may allocate the physical CPU 106A to operate with the logical partition 114A. Specifically, the hypervisor 112 may dispatch the logical CPU 116B on the physical CPU 106A.” (column 1, lines 33-43). Green does not disclose or teach the claimed scheduling scheme.

Bitar discloses the claimed scheduling scheme as a method for context switching between execution entities (abstract) wherein the execution entities may be virtual processors (column 1, lines 34-40; column 1, lines 55-63) but are equivalently described by Bitar as threads [“A virtual processor may be a process, [...] a kernel thread, [...] or some other abstraction.” (column 1, lines 34-40); “This abstraction, a virtual processor, is scheduled by the operating system scheduler for execution on available physical processors.” (column 1, lines 55-63)].

Art Unit: 2123

Bitar's method of switching between entities comprises a yielding virtual processor ["threads which have finished their work", (column 10, line 48 – column 11, line 10)] which designates a target virtual processor ["can transfer control of their respective processors to the preempted threads, thus resuming them." (column 10, line 48 – column 11, line 10); "thread B can transfer its processor to thread A, thus resuming thread A" (column 11, lines 33-41)] and switching-in the target virtual processor for execution by the CPU in response to the requested yield ["resuming thread them"; "resuming thread A"; supra].

It would have been obvious to a person of ordinary skill in the art at the time of Applicants' invention to combine the teachings of Bitar regarding designating a target virtual processor by a yielding virtual processor on a logically partitioned computer system using a hypervisor because Green expressly teaches that the hypervisor schedules virtual processors and Bitar teaches an improved scheduling algorithm for dealing with situations such as spin locks (as in column 11, lines 33-41). The combination could be formed by implementing the processor control transfer method taught by Bitar in the logically partitioned hypervisor system of Green.

The Examiner further relied upon the Silberschatz reference to provide evidence of the inherency using a virtual processor to schedule threads, as follows.

Silberschatz teaches on page 74:

This layered approach is taken to its logical conclusion in the concept of a *virtual machine*. The VM operating system for IBM systems is the best example of the virtual-machine concept, because IBM pioneered the work in this area.

By using CPU scheduling (Chapter 5) and virtual-memory techniques (Chapter 9), an operating system can create the illusion of multiple processes, each executing on its own processor with its own (virtual) memory. ... Each process is provided with a (virtual) copy of the underlying computer (Figure 3.12).

The resources of the physical computer are shared to create the virtual machines. CPU scheduling can be used to share the CPU and to create the appearance that users have their own processor.

It is noted that the assignee for this application is "International Business Machines Corporation," presumably the same "IBM" referred to by Silberschatz. If the virtual machine and integral virtual processor did not schedule threads, it would be impossible for that virtual machine to "create the appearance that users have their own processor," because scheduling threads is inherent to the operation of a processor. Even a single-threaded system must schedule the single thread to operate correctly.

Regarding claim 2, Bitar et al. teaches a method of context switching wherein the target virtual processor requires access to the CPU, wherein the yielding virtual processor controls the CPU (column 10, line 48 – column 11, line 10; column 11, lines 33-41).

Regarding claim 3, Bitar et al. teaches a method of context switching comprising generating a yield command from the virtual processor, wherein the yield command includes pointer and status information regarding the target virtual processor (column 10, lines 9-33).

Regarding claim 4, Bitar et al. teaches a method of context switching comprising assigning status information to the target virtual processor (column 10, lines 9-33).

Regarding claim 5, Bitar et al. teaches a method of context switching comprising assigning a target count to the target virtual processor (column 10, lines 9-33). The preempt bit

Art Unit: 2123

vector holds a value of 0 for a thread that has its resource requirements fulfilled and holds a value of 1 for a thread that has been preempted and requires resources to continue.

Regarding claim 6, Bitar et al. teaches a method of context switching comprising comparing the target count to a presented count conveyed in the yield request (column 10, lines 9-33; column 13, line 53 – column 14, line 24; column 16, lines 29-44).

Regarding claim 7, Bitar et al. teaches a method of context switching comprising aborting the yield in response to a yield-to-active command. If the processor is not needed, it will be reallocated to another process (column 16, lines 29-44).

Regarding claim 8, Bitar et al. teaches a method of context switching comprising designating the yielding virtual processor as waiting for the target virtual processor (column 10, line 48 – column 11, line 10; column 11, lines 33-41; column 16, lines 29-44).

Regarding claim 9, Bitar et al. teaches a method of context switching comprising designating the target virtual processor as having a yielding processor waiting for the target virtual processor (column 13, line 53 – column 14, line 24; column 16, lines 29-44).

Regarding claim 10, Bitar et al. teaches a method of context switching comprising storing the state of the yielding virtual processor (column 10, lines 9-33; column 13, lines 53-60).

Art Unit: 2123

Claims 11-18 are directed toward an apparatus comprising a computer system and a computer program to execute the method of claims 1-3, and 5-9. As the invention of Bitar et al. is a computer system and program (abstract), claims 11-18 are rejected for reasons similar to those given for claims 1-3, and 5-9 above.

Claim 19 is directed toward a program product and tangible signal bearing medium bearing a computer program which executes the method of claim 1. As the invention of Bitar et al. can be realized with a computer program, whether transmitted via a network or stored locally (abstract; column 17, lines 20-27; column 20, line 49 – column 21, line 10), claims 19 and 20 are rejected for reasons similar to those given for claim 1 above.

Regarding claim 21, Bitar teaches a user-level scheduler which includes a queue (schedule) used to schedule ready-to-run threads (column 8, lines 21-31). It is inherent that a user-level scheduler is a thread, therefore a virtual processor.

(10) Response to Argument

Appellant argues:

Equating a virtual processor to a thread is contrary to the known meaning of the terms, the express language of the primary reference, itself, and even the previous position of the Examiner. Moreover, such a misrepresentation is non sequitur in terms of the language of the presently pending claims. Because a virtual processor is patentably distinct from a thread, the prior art fails to suggest or motivate anything more than thread switching. (Brief, page 6 of 13)

The Examiner has fully considered this argument and finds it unpersuasive for the following reasons.

Art Unit: 2123

Applicants' specification, page 2, lines 15-16 states:

In this manner, virtual processors act as logical threads of execution for a host partition.

The *Bitar* reference, column 1, lines 33-36, states:

A virtual processor may be a process, such as that provided by traditional UNIX systems, a kernel thread, such as that provided by Mach, or some other abstraction.

The *Bitar* reference, column 1, lines 55-63, states:

Every operating system exports an abstraction that represents the basic unit of scheduling. Under UNIX, for example, the process is the fundamental abstraction that is scheduled; under Mach the kernel thread is the equivalent entity. This abstraction, a virtual processor, is scheduled by the operating system scheduler for execution on available physical processors. It is called a virtual processor because an application may treat it as a processing resource independent of whether it is "backed" by a physical processor.

The *Bitar* reference, column 1, line 64 – column 2, line 5, states:

A traditional UNIX process cannot execute in parallel on a multiprocessor precisely because a virtual processor is a process (a single virtual processor can only be scheduled onto a single physical processor); multiple processes can run concurrently, but if there is only a single runnable process all the processors but one will be idle. A Mach process having multiple kernel threads can run concurrently on multiple processors since the virtual processor is a kernel thread and the process may be comprised of multiple kernel threads.

Appellants' specification and the prior art of record refutes Appellants' allegation that a virtual processor and a thread are patentably distinct. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

To support his position, however, the Examiner erroneously asserts that threads are the equivalent of virtual processors. That is, the basis of Examiner's argument is that the thread switching processes of *Bitar* are the same as the claimed virtual processor steps. This position of equating a virtual processor to a thread is contrary to the known meaning

Art Unit: 2123

of the terms, the express language of the primary reference, itself, and even the previous position of the Examiner. (Brief, page 8 of 13)

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

The assertion of the Examiner is especially discouraging because *Bitar*, itself, dedicates more than a paragraph to distinguishing between threads and the claimed virtual processors: "Conceptually, it is useful to distinguish the user-level thread from the virtual processors" (col. 1, lines 27-33). Basically, *Bitar* explains this distinction as being between (user) threads ("the instantiation of each activity during execution" – col. 1, lines 16-19) and virtual processors ("the basic unit of scheduling" – col. 1, line 56). (Brief, page 8 of 13)

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above, in addition to the following.

The *Bitar* reference explicitly defines the term *virtual processor* as equivalent to *kernel thread* in several instances (column 1, lines 33-36; column 1, lines 56-61; column 2, lines 2-5). Therefore, where *Bitar* establishes a conceptual distinction between the *user-level* thread and the virtual processor, that distinction is equivalently between a *user-level* thread and a *kernel* thread. This distinction is not, as Appellants allege, between the generic concept of a thread and a virtual processor. Appellants' allegation is unsupported by the plain text of the *Bitar* reference, contradicts those portions where *Bitar* defines the term *virtual processor* as equivalent to *kernel thread* as cited above, and contradicts Appellants' specification as cited above.

As shown above, Appellants' specification defines a virtual processor as a logical thread of execution. The *Bitar* reference defines a virtual processor as equivalent to a kernel thread. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

Another instance in *Bitar* that plainly contradicts the Examiner's assertion that threads and virtual processor are equivalent includes Fig. 2b, which distinctly shows threads 2 mapped to a kernel space comprising virtual processors 5 (col. 5, lines 64-65). Fig. 8 likewise shows threads 24.M switched without using kernel space 18 (and associated virtual processors 45). (Brief, page 8 of 13)

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above, in addition to the following.

The Examiner respectfully submits that Appellants have misconstrued the teachings of the *Bitar* reference regarding the distinction between a *user-level* thread and the virtual processor or, equivalently, a *kernel* thread. The portions of the specification cited by Appellants may illustrate a *user-level* thread, but nothing in the *Bitar* reference supports Appellants' allegation that a virtual processor is patentably distinct from a thread.

As shown above, Appellants' specification defines a virtual processor as a logical thread of execution. The *Bitar* reference defines a virtual processor as equivalent to a kernel thread. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

In the context of *Bitar*, this distinction between user threads and virtual processors (or kernel threads in a Mach system, col. 1, lines 54-59 and col. 2, line 63) is important because *Bitar* is focused on switching threads in a way that does not burden virtual processors. The objective of *Bitar* is to achieve switching between user threads without involving the scheduling of virtual processors (col. 5, lines 31-33 and lines 55-58). *Bitar* teaches away from using a virtual processor, or kernel, and associated scheduling during thread switching for efficiency reasons (col. 5, lines 14-18, 31-38, 55-58, and col. 12, line 24). (Brief, pages 8-9 of 13)

Art Unit: 2123

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above, in addition to the following.

The *Bitar* reference, abstract, states:

A system and method for context switching between a first and a second execution entity (such as a thread) without having to enter into protected kernel mode.

The *Bitar* reference clearly defines that “a virtual processor is a kernel thread” (column 1, lines 33-36; column 2, lines 2-5).

Therefore, the distinction in *Bitar* between a *user-level* thread and a virtual processor, or equivalently a *kernel* thread, is not understood to be particularly relevant. Neither Appellants’ specification nor the *Bitar* reference supports Appellants’ allegation that a virtual processor is patentably distinct from a thread. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

Perhaps the Examiner’s confusion has arisen from *Bitar* text that describes a *kernel* thread in a Mach operating system as being the equivalent of a virtual processor in other operating systems (column 1, lines 54 – column 2, line 5). However, *Bitar* characterizes both the kernel thread and the virtual processor as being “the basic unit of scheduling”. What these units are being used to schedule, or map, are *user* threads. Of note, claim 1 explicitly recites the function of the virtual processor as a entity used for scheduling. Applicants consequently submit that the functional distinction between a virtual processor and a thread is patentably clear. (Brief, page 9 of 13)

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above, in addition to the following.

The Examiner agrees with Appellants’ argument that *Bitar* teaches kernel threads, or equivalently virtual processors, being used to schedule, or map, *user* threads. Claim 1 recites, “a

Art Unit: 2123

plurality of virtual processors used to schedule threads”. The *Bitar* reference appears to teach that limitation.

Applicants’ specification, page 2, lines 15-16 states:

In this manner, virtual processors act as logical threads of execution for a host partition.

The *Bitar* reference, column 1, lines 33-36, states:

A virtual processor may be a process, such as that provided by traditional UNIX systems, a kernel thread, such as that provided by Mach, or some other abstraction.

The Examiner respectfully submits that the functional distinction between a virtual processor and a thread is *not* patentably clear. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

As part of his basis for asserting that a virtual processor can be a user thread, the Examiner asserts that an “execution entity” can be a virtual processor in terms of the disclosure of *Bitar*. As discussed above, however, *Bitar* describes only threads as execution entities, in contrast with scheduling entities, such as virtual processors (column 1, lines 16-19 and 56). As such, Examiner’s attempt to expand *Bitar*’s definition of execution entities to include virtual processors is improper and contrary to the plain text of *Bitar*.

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above, in addition to the following.

The Examiner does not assert that a virtual processor can be a *user* thread. The Examiner merely observes that Appellants’ specification, page 2, lines 15-16 states:

In this manner, virtual processors act as logical threads of execution for a host partition.

The *Bitar* reference, column 1, lines 33-36, states:

Art Unit: 2123

A virtual processor may be a process, such as that provided by traditional UNIX systems, a kernel thread, such as that provided by Mach, or some other abstraction.

The Examiner does not “expand” *Bitar*’s definition of execution entities. The Examiner has interpreted Appellants’ claim according the specification and has interpreted the prior art according to the explicit language found therein.

Appellant argues:

In any case, as presently claimed, virtual processors are different than threads. The express claim language recites the function of a virtual processor for use in scheduling thread execution. Conventional operating system dispatch threads (units of execution) to CPU’s. In a logically partitioned system, the operating system thinks that the virtual processors are CPU’s for purposes of dispatching threads. That is, the operating system dispatches threads to virtual processors. The hypervisor schedules CPU resources to the virtual processors to execute the threads dispatched to the virtual processors. The virtual processor is used (by the hypervisor) to schedule thread execution. As such, the Examiner’s assertion that a virtual processor is equivalent to a thread is contrary to the known meaning of the words, the express claim language, and the terms as defined in the cited prior art.

The Examiner has fully considered this argument and finds it unpersuasive for the reasons shown above. In summary, Appellants’ specification, page 2, lines 15-16 states:

In this manner, virtual processors act as logical threads of execution for a host partition.

The *Bitar* reference, column 1, lines 33-36, states:

A virtual processor may be a process, such as that provided by traditional UNIX systems, a kernel thread, such as that provided by Mach, or some other abstraction.

The *Bitar* reference, column 1, lines 55-63, states:

Every operating system exports an abstraction that represents the basic unit of scheduling. Under UNIX, for example, the process is the fundamental abstraction that is scheduled; under Mach the kernel thread is the equivalent entity. This abstraction, a virtual processor, is scheduled by the operating system scheduler for execution on available physical processors. It is called a virtual processor because an application may

Art Unit: 2123

treat it as a processing resource independent of whether it is “backed” by a physical processor.

The *Bitar* reference, column 1, line 64 – column 2, line 5, states:

A traditional UNIX process cannot execute in parallel on a multiprocessor precisely because a virtual processor is a process (a single virtual processor can only be scheduled onto a single physical processor); multiple processes can run concurrently, but if there is only a single runnable process all the processors but one will be idle. A Mach process having multiple kernel threads can run concurrently on multiple processors since the virtual processor is a kernel thread and the process may be comprised of multiple kernel threads.

Appellants’ alleged distinction between a virtual processor and a thread is supported by neither Appellants’ specification nor the prior art of record. The Examiner maintains that a virtual processor is a thread.

Appellant argues:

Appellants’ arguments directed to claims 2-19 and 21 all refer back to the alleged deficiencies addressed above. These arguments have been fully considered but have been found unpersuasive, as explained above.

(11) Related Proceeding(s) Appendix

No decision rendered by a court or the Board is identified by the examiner in the Related Appeals and Interferences section of this examiner’s answer.


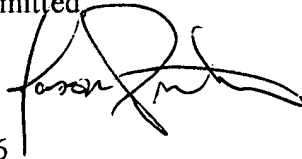
Art Unit: 2123

For the above reasons, it is believed that the rejections should be sustained.

Respectfully submitted

Jason Proctor

October 26, 2006



PAUL RODRIGUEZ
SUPERVISORY PATENT EXAMINER
TECHNOLOGY CENTER 2100

10/30/06

Conferees:

Anthony Knight



Paul Rodriguez

